# PROGRAMACIÓN EN MIDP

Celeste Campo
**celeste@it.uc3m.es**

**Parte de este material se basa en transparencias de Natividad Martínez Madrid (nati@it.uc3m.es)**



# ÍNDICE


- MIDlet.
- Primer ejemplo: “Hola Mundo”
- Interfaz gráfico (UI):
  - API UI de alto nivel
  - API UI de bajo nivel
- RMS: Record Management System
- GCF: Generic Connection Framework


# BIBLIOGRAFÍA


- *Wireless Java Programming with Java 2 Micro Edition*. Feng, Yu and Zhu, Jun. SAMS [2001]. L/D 004.438 JAVA FEN. Capítulos 5 al 9.
- *J2ME : Java 2 micro edition : manual de usuario y tutorial*. Froufe, Agustín y Jorge, Patricia. Ra-Ma. [2004]. L/S 004.438 JAVA FRO, L/D 004.438 JAVA FRO.

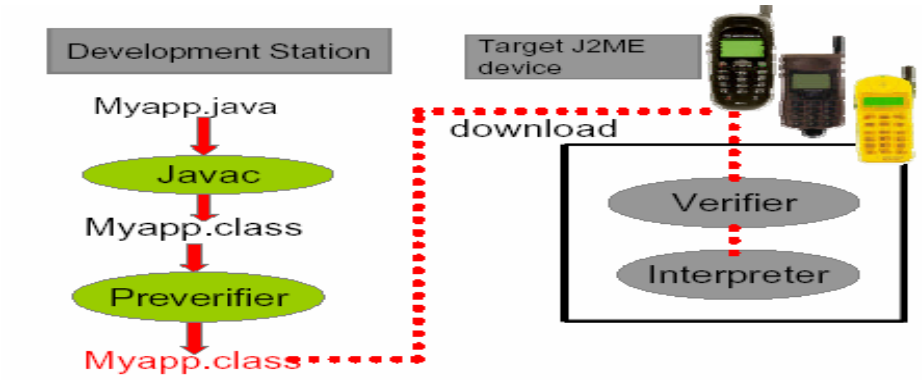
# MIDlet – Desarrollo y despliegue

- MIDlet: aplicación MIDP.
- El desarrollo de un MIDlet consta de las siguientes etapas:
  - Crear.
  - Publicar.
  - Descargar.
  - Instalar.
  - Ejecutar.
  - Actualizar (gestión de versiones).
  - Borrar.

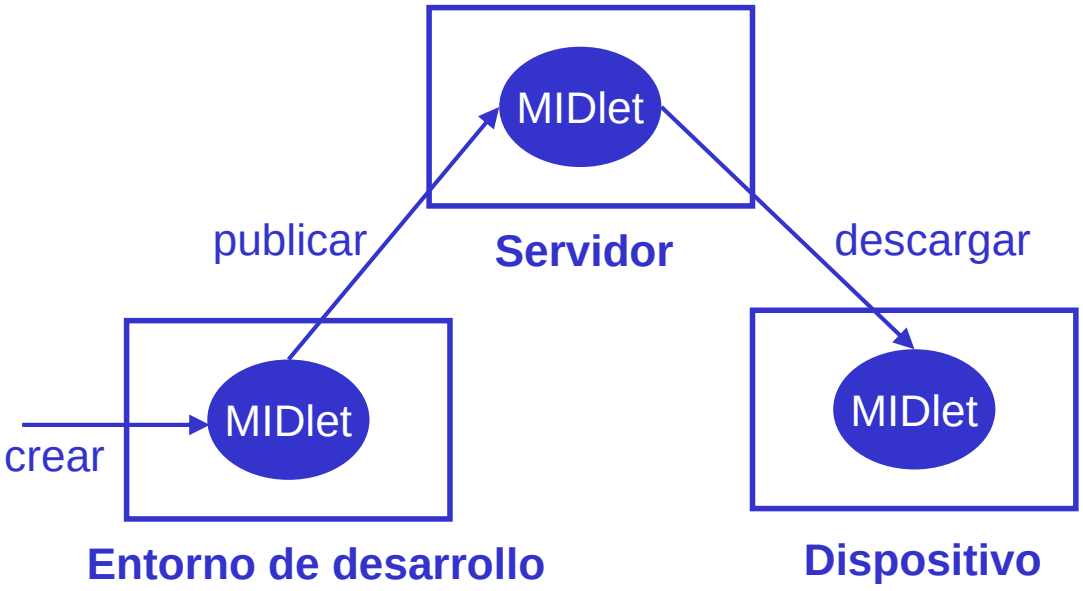


# MIDlet - Creación

- Los pasos de creación de un MIDlet:
  - Escribir el código y compilar.
  - Preverificar el código.
  - Empaquetar en un JAR y crear el descriptor (fichero JAD).
  - Ejecutar en emulador.



# MIDlet - Publicación






# MIDlet - Descarga

- Gestionado por el *Application Management System*.
- El dispositivo obtiene el MIDlet de alguna fuente, a través:
  - Red inalámbrica.
  - Puerto serie.
  - IRDA.
  - Bluetooth.
  - ...
- Negociación sobre capacidades del dispositivo según los requisitos del MIDlet, coste...
- Se descarga el MIDlet a la memoria del dispositivo.

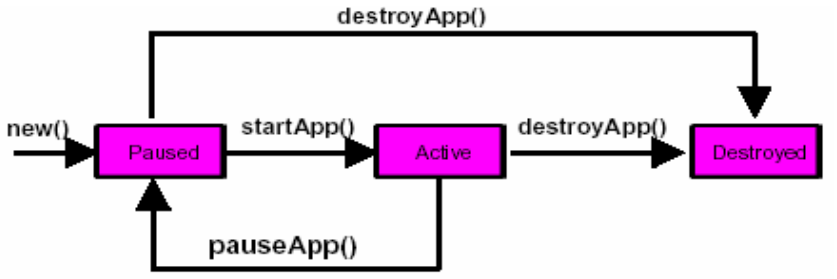
# MIDlet - Instalación

- Gestionado por el *Application Management System*.
- Puede comprobar que el MIDlet no vulnera las políticas de seguridad del móvil.
- Puede transformar (convertir) el MIDlet de formato "público" a un formato específico del dispositivo:
  - Ejemplo: en PalmOS se transforma a formato PRC.



# MIDlet - Ejecución

- El usuario selecciona el MIDlet y lo ejecuta.
- En este momento, el MIDlet entra en la VM y se invocan los métodos que gestionan su ciclo de vida:
  - **Paused**: Iniciado y a la espera.
  - **Active**: Tiene recursos ejecutando.
  - **Destroyed**: Ha liberado recursos, destruido hilos y terminado toda su actividad.



# MIDlet - Actualización

- Puede publicarse una nueva versión del MIDlet.
- *Application Management System* debe gestionar la lista de MIDlets instalados y sus versiones.
  - Puede así actualizar de versiones más antiguas a más recientes del MIDlet.
- Los atributos del MIDlet, incluida la versión, están:
  - En el descriptor del MIDlet (JAD).
  - En el manifiesto del MIDlet contenido en el JAR.



# MIDlet - Borrado

- A través del *Application Management System* puede borrarse un MIDlet cuando no va a utilizarse más.
- Se borra:
  - MIDlet.
  - Todos los registros en memoria permanente escritos por ese MIDlet.

# MIDlet – JAR y Manifiesto

- Incluye los ficheros de clases y otros recursos asociados al MIDlet, por ejemplo imágenes.
- Fichero JAR (formato estándar) puede contener varios MIDlets, es lo que se denomina MIDlet Suite.
- El manifiesto está incluido en el JAR y contiene información sobre los contenidos del fichero JAR:

| Atributos obligatorios | Atributos opcionales |
|---|---|
| `MIDlet-Name` | `MIDlet-Description` |
| `MIDlet-Version` | `MIDlet-Icon` |
| `MIDlet-Vendor` | `MIDlet-Info-URL` |
| `MIDlet-<n> (name, icon, class)` | `MIDlet-Data-Size` |
| `MicroEdition-Profile` | `MIDlet-Jar-Size` |
| `MicroEdition-Configuration` | `MIDlet-Jar-URL` |



# MIDlet – Descriptor (JAD)

- Permite que el *Application Management System* verifique si el MIDlet es indicado antes de descargarlo.
- Es un fichero de texto con extensión **`.jad`**.
- Atributos requeridos:
  - **`MIDlet-Name`**
  - **`MIDlet-Version`**
  - **`MIDlet-Vendor`**
  - **`MIDlet-Jar-URL`**
  - **`MIDlet-Jar-Size`**
  - **`MicroEdition-Profile`**
  - **`MicroEdition-Configuration`**
- Atributos opcionales:
  - **`MIDlet-Description`**
  - **`MIDlet-Icon`**
  - **`MIDlet-Info-URL`**
  - **`MIDlet-Data-Size`**
- Puede incluir otros atributos que no necesariamente son “MIDlet-”

# MIDlet - `javax.microedition.midlet.MIDlet`

- Clase abstracta base para todos los MIDlets:
  - **`protected MIDlet()`**
    - Constructor para crear un MIDlet, sin argumentos.
  - **`protected abstract void startApp() throws MIDletStateChangeException`**
    - Se le llama cuando el MIDlet comienza (paused -> active state)
  - **`protected abstract void pauseApp()`**
    - Se le llama cuando el MIDlet pausa (active -> paused state)
  - **`protected abstract void destroyApp(boolean unconditional)throws MIDletStateChangeException`**
    - Se le llama cuando el MIDlet se destruye (-> destroyed state)
  - **`public final void notifyDestroyed()`**
    - Comunica al AMS que el MIDlet ha limpiado la memoria y ha terminado.
  - **`public final void notifyPaused()`**
    - Comunica al AMS que el MIDlet está en pausa.
  - **`public final String getAppProperty(String key)`**
    - Se le llama para obtener las propiedades del MIDlet.



# MIDlet – MIDlet Suite

- MIDlet Suite:
  - Conjunto de MIDlets contenidas en un mismo .JAR.
- Normalmente aunque sólo se incluya un MIDlet se trabaja con MIDlet Suite.
- Los MIDlets en un MIDlet Suite comparten recursos:
  - Contenidos en el JAR.
  - RMS creados por los otros MIDlets (en el mismo MIDlet Suite).

# “Hola Mundo” en MIDP

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet implements CommandListener{

  private Display display;
  private TextBox mainScreen = null;
  private Command exit = new Command("exit", Command.EXIT, 2);

  public HelloWorld() {
    display = Display.getDisplay(this);
    mainScreen = new TextBox("Text Box", "Hola alumnos de SWC",
                             512,0);
    mainScreen.addCommand(exit);
    mainScreen.setCommandListener(this);
  }
```



# “Hola Mundo” en MIDP

```
  public void startApp() {
    display.setCurrent(mainScreen);
  }
  public void pauseApp() {
  }
  public void destroyApp(boolean unconditional) {
  }
  public void commandAction(Command c, Displayable s) {
    if (c == exit) {
      destroyApp(false);
      notifyDestroyed();
    }
  }
}
```

# “Hola Mundo” en MIDP

- Fichero .JAD

```
MIDlet-1: HelloWorld, HelloWorld.png, HelloWorld
MIDlet-Jar-Size: 1183
MIDlet-Jar-URL: HelloWorld.jar
MIDlet-Name: HelloWorld
MIDlet-Vendor: Sun Microsystems
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```



# “Hola Mundo” en MIDP

- Manifiesto

```
MIDlet-1: HelloWorld, HelloWorld.png, HelloWorld
MIDlet-Name: HelloWorld
MIDlet-Vendor: Sun Microsystems
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

# “Hola Mundo” en MIDP

Ver código: **`HelloWorld.java`** – **`HelloWorld.zip`** (proyecto WTK)



# Explicación de “Hola Mundo” en MIDP

- Extiende MIDlet:
  - Implementa **`startApp()`**, **`pauseApp()`**, **`destroyApp()`**
- Atributos privados:
  - **`Display display`**: representa el gestor de la pantalla y de los dispositivos de entrada. Uno por MIDlet.
  - **`TextBox mainScreen`**: permite al usuario introducir y editar texto `display.setCurrent(mainScreen)` (en `startApp()` hace el `TextBox` visible.
  - **`Command exit`**: representa apretar una tecla
    - Asociado a la mainScreen: **`mainScreen.addCommand(exit)`**
    - El MIDlet se registra en mainScreen como escuchador de exit: **`mainScreen.setCommandListener(this)`**
    - El MIDlet implementa el método escuchador **`commandAction()`** (interfaz **`CommandListener`**)

# Interfaz gráfico

- Paquete:
  - **`javax.microedition.lcdui`**
- Clases básicas:
  - **`Displayable`**:
    - Información a ser visualizada.
  - **`Display`**:
    - Selecciona qué objeto `Displayable` se muestra al usuario.



# Interfaz gráfico – Clase `Displayable`

- Existen tres categorías de Displayables:
  - Screen con estructura predefinida:
    - **`Alert`**, **`List`** o **`TextBox`**
    - Encapsulan componentes de interfaces complejos que las aplicaciones no pueden enriquecer con nuevos componentes.
  - Screen genérico:
    - **`Form`**
    - Las aplicaciones pueden llenar esta pantalla con texto, imágenes u otros componentes de interfaz gráfico.
  - Canvas (displayable de bajo):
    - Usuario tiene control total sobre los componentes del display y puede acceder a eventos de bajo nivel.

# Interfaz gráfico – Clase `Display`

- Métodos para controlar la visualización de objetos **`Displayable`** y obtener propiedades del display (num. colores, etc).
- Sólo uno por MIDlet (*singleton*).
- Obtención del objeto Display:
  - **`static Display Display.getDisplay(MIDlet m)`**
- Obtener el **`Displayable`** que se está visualizando:
  - **`Displayable getCurrent()`**
- Establecer el **`Displayable`** a visualizar (current)
  - **`void setCurrent(Displayable nextDisplayable)`**
  - **`void setCurrent(Alert alert, Displayable nextDisplayable)`**



# Interfaz gráfico – Eventos y su gestión

- Mismo modelo que AWT:
  - Fuentes de eventos y escuchadores (listeners) de evento
- Gestión de eventos en el mismo thread en el que se produce el evento.
- Eventos de alto nivel:
  - **`Command`**:
    - `CommandListener`
    - Fuentes: `Displayable`
  - **`ItemStateChanged`**
    - `ItemStateListener`
    - Fuentes: `Form`
- Eventos de bajo nivel:
  - Relacionados con pulsaciones de teclas, de puntero,…
  - Una aplicación no puede acceder a eventos de bajo nivel a través de displayables de alto nivel.

# Interfaz gráfico – Clase `Command`

- Un objeto **`command`** contiene tres elementos:
  1. Label:
     - String representando el significado del comando.
     - Es lo que la aplicación muestra al usuario.
  2. Type:
     - Entero que especifica qué va a realizar el comando.
     - Tipos definidos: `BACK`, `CANCEL`, `HELP`, `EXIT`, `ITEM`, `OK`, `SCREEN` y `STOP`.
  3. Priority:
     - Entero que indica la importancia del comando.
     - Mayor cuanto menor sea el número.
- Constructor:
  **`Command(String label, int commandType, int priority)`**
- Añadir comando a un displayable:
  **`addCommand(Command)`**
- Eliminar comando de un displayable:
  **`removeCommand(Command)`**
- CommandListener (interfaz):
  - Método **`commandAction (Comand c, Displayable d)`**



# API UI de alto nivel – Clase `List`

- Subclase de Screen que implementa el interfaz **`Choice`**
- Constructores:
  - **`List(String title, int listType):`**
    - Se crea una lista vacía, los elementos de selección se pueden añadir después.
  - **`List(String title, int listType, String[] stringElements,Image[] imageElements:`**
    - Se indican los elementos de la lista y posibles imágenes asociadas a cada elemento.

# API UI de alto nivel – Interfaz `Choice`

- Presenta al usuario una serie de elementos (texto y posiblemente imagenes) que puede seleccionar.
- Métodos:
  - **`int size()`**: número de elementos de selección.
  - **`void set(int elementNum, String stringPart, Image imagePart)`**: modifica elemento indicado.
  - **`String getString(int elementNum)`**: obtiene texto del elemento.
  - **`Image getImage(int elementNum)`**: obtiene imagen del elemento.
  - **`void setSelectedFlags(boolean[] selectedArray)`**: modificar estado de selección de todos los elementos.
  - **`void setSelectedIndex(int elementNum, boolean selected)`**: idem de un elemento concreto.
  - **`void insert(int elementNum, String stringPart, Image imagePart)`**: inserta elemento en posición previa al elemento indicado.
  - **`int append(String stringPart, Image imagePart)`**: añade elemento.



# API UI de alto nivel – Interfaz `Choice`

- Existen tres tipos de objetos Choice:
  1. Choice exclusivo:
     - Exactamente un elemento seleccionado simultáneamente.
  2. Choice implícito:
     - Sólo se puede utilizar en listas (`List`), donde el elemento que está “enfocado” es el que se selecciona implícitamente.
  3. Choice multiple:
     - Se puede seleccionar cualquier número de elementos, incluido ninguno, y en cualquier combinación.

# API UI de alto nivel – Ejemplos `List` y `Choice`

Ver código: **`ListTest.java`** – **`ChoiceExample.zip`** (proyecto WTK)



# API UI de alto nivel – Clase `TextBox`

- Subclase de Screen que permite al usuario introducir y editar un texto.
- Constructor
  - **`TextBox(String title, String text, int maxSize, int constraints)`**
- Tamaño máximo:
  - Indicado en el constructor y se puede modificar con **`setMaxSize`**.
- Restricciones de entrada:
  - **`ANY`**: cualquier texto
  - **`EMAILADDR`**: dirección email
  - **`NUMERIC`**: valor entero
  - **`PHONENUMBER`**: número de teléfono
  - **`URL`**: URL

# API UI de alto nivel – Clase `TextBox`

- Métodos para editar y obtener el texto del **`TextBox`**:
  - Contenido como array de caracteres:
    - **`void setChars(char[] data, int offset, int legth)`**
    - **`int getChars(char[] data)`**
  - Contenido como String:
    - **`void setString(String text)`**
    - **`String getString()`**
  - Modificar el contenido a partir de una posición:
    - **`void insert(char[] data, int offset, int length, int position)`**
    - **`void insert(String src, int position)`**
  - Borrar el contenido:
    - **`void delete(int offset,int length)`**
- A través del teclado.



# API UI de alto nivel – Clase `Alert`

- Subclase de Screen que permite visualizar datos (texto, imágenes) durante un cierto tiempo (timeout) antes de pasar a otra pantalla.
- Constructores:
  - **`Alert(String title)`**
  - **`Alert(String title, String alertText, Image alertImage, AlertType alertType)`**
- Timeout en milisegundos:
  - Establecido con el método: **`setTimeout(int time)`**
  - Temporizador infinito (**`Alert.FOREVER`**): cambio de pantalla mediante comando.
- Tipos de alertas
  - **`ALARM`**, **`CONFIRMATION`**, **`ERROR`**, **`INFO`**, **`WARNING`**
- Siguiente displayable que se presenta después del timeout:
  - **`display.setCurrent(Alert alert, Displayable displayable)`**

# API UI de alto nivel – Clase `Form`

- Subclase de Screen que puede contener un número arbitrario de items
- Constructores:
  - **`Form(String title)`**
  - **`Form(String title, Item[] items)`**
- Métodos:
  - **`int append(Image img)`**
  - **`int append(Item item)`**
  - **`int append(String str)`**
  - **`void delete(int itemNum)`**
  - **`void insert(int itemNum, Item item)`**
  - **`void set(int itemNum, Item item)`**
  - **`Item get(int itemNum)`**
- Un item sólo puede colocarse en un Form



# API UI de alto nivel – Clase `Item`

- Conjunto de elementos que pueden añadirse a un Form, con una etiqueta asociada.
- Superclase de:
  - **`ChoiceGroup`**:
    - Item que implementa interfaz Choice (no implícita)
    - Parecido a **`List`**.
  - **`StringItem`**:
    - Visualizar texto (sólo las aplicaciones interaccionan, no el usuario)
  - **`ImageItem`**:
    - Con parámetro layout para alineación
  - **`TextField`**:
    - Texto que los usuarios pueden editar
  - **`DataField`**:
    - Editable para visualizar fechas y tiempos
  - **`Gauge`**:
    - Diagrama de barras

# API UI de alto nivel – Otras clases

- Clase **`Ticker`**:
  - Texto que pasa continuamente a través de la pantalla.
  - Se asociado a un displayable con el método: **`setTicker(Ticker ticker)`**.
- Clase **`Image:`**
  - Imágenes inmutables:
    - Generadas a partir de un fichero (recurso, descargado,…)
    - No se pueden modificar.
    - Tipo de imágenes en un **`Alert`**, **`Choice`** o **`ImageItem`**
  - Imágenes mutables:
    - Se pueden modificar.
- Clase **`CustomItem:`**
  - Clase abstracta que permite definir nuevos tipos de Items.
  - Clase MIDP 2.0.



# API UI de bajo nivel

- Control gráfico de la pantalla a nivel de pixel.
- Definición de eventos propios de entrada:
  - Detección de eventos de teclado.
  - Detección de eventos de puntero.
- Más flexibilidad, menos portabilidad:
  - El resultado final puede depender del teléfono concreto en el que se ejecute.
- Clase fundamental para la realización de juegos:
  - Mejorada en MIDP 2.0 con la extensión **`GameCanvas`**.

# API UI de bajo nivel – Clase Canvas

- Subclase abstracta de Displayable que permite realizar interfaces gráficos de bajo nivel en MIDP.
- Necesario obtener el tamaño del display y programar teniendo en cuenta estas dimensiones:
  - **`int getWidth()`**, **`int getHeight()`**
- Método **`void paint(Graphics g):`**
  - Debe pintar todos los pixeles de la pantalla.
- Métodos para la gestión de eventos a bajo nivel
  - Entradas teclado.
  - Puntero pantalla táctil.
  - Cuando se visualiza el Canvas en el display:
    - **`showNotify:`** antes de visualizarlo.
    - **`hideNotify:`** después de visualizarlo.



# API UI de bajo nivel – Clase Canvas

- Una clase que extienda Canvas:
  - Debe implementar obligatoriamente el método (abstracto) **`paint`**.
  - No es necesario que implemente todos los métodos relacionados con eventos a bajo nivel:
    - No son métodos abstractos y su implementación por defecto es vacía (no hacen nada).
    - El desarrollador sólo debe implementar los métodos correspondientes a los eventos que quiere gestionar.

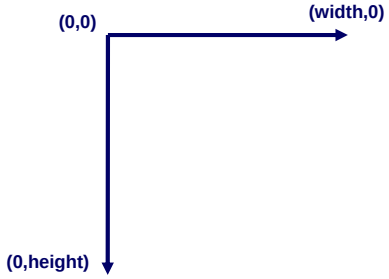
# API UI de bajo nivel – Clase `Graphics`

- **`Graphics`**:
  - Similar a `java.awt.Graphics`: geometría bidimensional
- Pasado al método **`paint`** de **`Canvas`**
- Métodos para dibujar:
  - `void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`
  - `void drawLine(int x1, int y1, int x2, int y2)`
  - `void drawRect(int x, int y, int width, int height)`
  - `void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`
  - `void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)`
  - `void fillRect(int x, int y, int width, int height)`
  - `void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`



# API UI de bajo nivel – Clase `Graphics`

- Método para pintar imágenes:
  - `void drawImage(Image img, int x, int y, int anchor)`
- Métodos de soporte a colores:
  - `int getBlueComponent()`
  - `int getColor()`
  - `int getGrayScale()`
  - `int getGreenComponent()`
  - `int getRedComponent()`
  - `void setColor(int RGB)`
  - `void setColor(int red, int green, int blue)`
  - `void setGrayScale(int value)`
- Sistema de coordenadas empieza en el extremo superior derecho.



# API UI de bajo nivel – Ejemplo `Graphics` y `Canvas`

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class LineTest extends MIDlet {
  private Display display;
  public LineTest() {
    display=Display.getDisplay(this);
  }
  public void startApp() throws MIDletStateChangeException {
    display.setCurrent(new LineTestCanvas());
  }
  public void pauseApp() {
  }
  public void destroyApp(boolean unconditional) {
  }
```



# API UI de bajo nivel – Ejemplo `Graphics` y `Canvas`

```
  class LineTestCanvas extends Canvas {
    public void paint(Graphics g) {
      // Tamaño del área de dibujo
      int width=this.getWidth();
      int height=this.getHeight();
      // Fondo de la pantalla blanco
      g.setColor(0xFFFFFF);
      g.fillRect(0,0,width,height);
      // Líneas en negro
      g.setColor(0x000000);
      // Dibujamos dos líneas (ejes)
      g.drawLine(0,height,width,0);
      g.drawLine(0,0,width,height);
    }
  }
}
```

# API UI de bajo nivel – Ejemplo `Graphics` y `Canvas`

Ver código: **`LineTest.java`** – **`CanvasGraphicExample.zip`** (proyecto WTK)



# API UI de bajo nivel – Eventos

- Eventos de teclado:
  - Cada tecla está definida por un código **`KEY_NUM1`**,…**`KEY_NUM9`**, **`KEY_STAR`** (*) y **`KEY_POUND`** (#).
  - Métodos que gestionan los eventos:
    - **`void keyPressed(int keyCode)`**
    - **`void keyReleased(int keyCode)`**
    - **`void keyRepeated(int keyCode)`**
- Acciones de juego:
  - Se definen una serie de códigos: **`UP`**, **`DOWN`**, **`RIGHT`**, **`LEFT`**, **`FIRE`**, **`GAME_A`**, **`GAME_B`**, **`GAME_C`** y **`GAME_D`**.
  - Mapeo a teclas con los métodos: **`getKeyCode(int gameAction)`** y **`getGameAction(int keyCode)`**
- Eventos de puntero:
  - Métodos que gestionan los eventos:
    - **`void pointerPressed(int x, int y)`**
    - **`void pointerReleased(int x, int y)`**
    - **`void pointerRepeated(int x, int y)`**

# RMS: Record Management System

- Paquete:
  - **`javax.microedition.rms`**
- Base de datos sencilla orientada a registro:
  - Permite almacenar datos de las aplicaciones de forma persistente.
  - Independiente del sistema de almacenamiento concreto del dispositivo.



# RMS – `RecordStore`

- Un *record store* (clase **`RecordStore`**) es una colección de *records*
  - Un *record* es un array de bytes de tamaño variable.
- Reglas para los RecordStore:
  - Nombre:
    - Combinación de 32 caracteres (case sensitive).
    - Debe ser único en un mismo MIDlet Suite.
  - RecordStores del mismo MIDlet Suite:
    - Se almacenan en el mismo espacio de nombres.
    - Permite que se compartan entre los MIDles pertenecientes al MIDlet Suite.
    - No accesibles para MIDlets de otros MidletSuite.

# RMS – Operaciones sobre `RecordStore`

- Crear/abrir:
  - **`static RecordStore openRecordStore (String recordStoreName, boolean createIfNecessary)`**
- Cerrar:
  - **`void closeRecordStore()`**
  - Siempre después de utilizarse debe cerrarse.
- Borrar:
  - **`static void deleteRecordStore(String recordStoreName)`**
  - Antes de borrar siempre se debe cerrar.



# RMS – Operaciones sobre `RecordStore`

- Los records se identifican unívocamente mediante el identificador (**`recordID`** de tipo **`int`**).
- Manipular records:
  - **`int addRecord(byte[] data, int offset, int numBytes)`**
  - **`void deleteRecord(int recordId)`**
  - **`byte[]getRecord(int recordId)`** y **`int getRecord(int recordId, byte[] buffer, int offset)`**
  - **`void setRecord(int recordId, byte[] newData, int offset, int numBytes)`**

# RMS – Cabecera `RecordStore`

- La cabecera de un RecordStore proporciona la siguiente información:
  - Número de records en el RecordStore:
    - **`int getNumRecords()`**
  - Número de versión:
    - **`int getVersion()`**
  - Momento de la última modificación:
    - **`long getLastModified()`**
  - Identificador del siguiente recordID (siguiente record que será añadido):
    - **`getNextRecordID()`**



# RMS – Interfaz `RecordListener`

- Monitorizar cambios en RecordStores.
- Gestión de eventos mediante los métodos:
  - **`void recordAdded(RecordStore recordStore, int recordId)`**
  - **`void recordChanged(RecordStore recordStore, int recordId)`**
  - **`void recordDeleted(RecordStore recordStore, int recordId)`**
- Para añadir y borrar listeners:
  - **`void addRecordListener(RecordListener listener)`**
  - **`void removeRecordListener(RecordListener listener)`**

# RMS – Interfaz `RecordEnumeration`

- Después de borrar records sus identificadores ya no son consecutivos.
- Para recorrerlos se proporciona la clase **`RecordEnumeration`**:
  - Lista doblemente enlazada en la que cada nodo representa un record.
  - Se obtiene a través del método:
    - **`RecordEnumeration enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated)`**
  - Métodos:
    - **`void reset()`**: puntero al primer elemento de la lista.
    - **`int nextRecordId()`**: ID del siguiente elemento de la lista.
    - **`int previousRecordId()`**: ID del anterior elemento de la lista.
  - Se define como interfaz pero los fabricantes deben realizar una implementación de ella: para los desarrolladores es una clase.



# RMS – Interfaz `RecordFilter` y `RecordComparator`

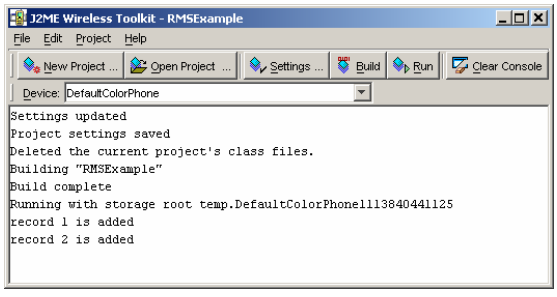
- Permiten ordenar los records en un RecordStore según algún criterio.
- Interfaz **`RecordFilter`**:
  - **`boolean matches(byte[] candidate)`**
  - La aplicación determina si el record (**`candidate`**) verifica el criterio de selección.
- Interfaz **`RecordComparator`**:
  - **`int compare(byte[] rec1, byte[] rec2)`**
  - Clasificación de records por algún criterio (ej. fecha creación)
  - Devuelve el orden de records (**`rec1`** y **`rec2`**):
    - **`PRECEDES`**, **`FOLLOWS`**, **`EQUIVALENT`**.

# RMS – Ejemplo

```
public void startApp() throws MIDletStateChangeException {
   RecordStore rs = null;
   try {
     rs = RecordStore.openRecordStore("file1", true);
     byte data[] = new byte[4];
     for ( int j=0; j<2; j++) {
        int i = rs.getNextRecordID();
        data[0] = (byte)((i >> 24) & 0xff);
        data[1] = (byte)((i >> 16) & 0xff);
        data[2] = (byte)((i >> 8) & 0xff);
        data[3] = (byte)(i & 0xff);
        System.out.println("record " + rs.addRecord(data,0,4) +
                           " is added");
     }
   } catch (Exception e) {}
   finally{
     try {
    rs.closeRecordStore();
     } catch (Exception e) {}
   }
   destroyApp(true);
   notifyDestroyed();
 }
```



# RMS – Ejemplo


J2ME Wireless Toolkit - RMSExample
File Edit Project Help
New Project ... Open Project ... Settings ... Build Run Clear Console
Device: DefaultColorPhone
Settings updated
Project settings saved
Deleted the current project's class files.
Building "RMSExample"
Build complete
Running with storage root temp.DefaultColorPhone1113840441125
record 1 is added
record 2 is added

Ver código: **`RecordStoreTest.java`** – **`RMSExample.zip`** (proyecto WTK)

# GCF: Generic Connection Framework

- Paquete:
  - **`javax.microedition.io`**
- Marco genérico de conexiones:
  - Interfaces definidas a nivel CLDC.
  - Implementadas en los perfiles y además, los perfiles pueden definir más interfaces:
    - En MIDP sólo es obligatorio implementar conexiones HTTP.



# GCF – Interfaces de conexión en CLDC

- **`Connection.`**
- **`InputConnection`**:
  - Representa un dispositivo desde el que se leen datos.
  - Método **`openInputStream`**
- **`OutputConnection`**:
  - Representa un dispositivo en el que se escriben datos.
  - Método **`openOutputStream`**
- **`StreamConnection`**
  - Combina las dos anteriores.
- **`ContentConnection`**:
  - Acceso a algunos metadatos de conexión HTTP.
- **`StreamConnectionNotified`**
  - Espera a que se establezca una conexión, devuelve un **`StreamConnection`**
- **`DatagramConnection`**:
  - Envío y recepción de datagramas.

# GCF – Interfaces de conexión en MIDP

- **`CommConnection`**: extiende de **`StreamConnection`**.
  - Conexión a puerto serie (**`comm:`**).
- **`HttpConnection`**: extiende **`ContentConnection`**.
  - Conexión HTTP (**`http:`**).
- **`HttpsConnection`**: extiende **`HttpConnection`**.
  - Conexión HTTP segura (**`https`**:).
- **`SocketConnection`**: extiende de StreamConnection.
  - Socket TCP/IP (**`socket:`**)
- **`ServerSocketConnection`**: extiende de **`StreamConnectionNotifier`**.
  - Socket pasivo TCP/IP (**`socket:`**)
- **`SecureConnection`**: extiende de **`SocketConnection:`**
  - Socket seguro (**`ssl:`**)
- **`UDPDatagramConnection`**: extiende de **`DatagramConnection`**.
  - Conexión de datagramas TCP/IP (**`datagram:`**)



# GCF – Clase `Connector`

- Método open:
  - Conexión HTTP:
    - **`Connector.open("http://www.it.uc3m.es/")`**
  - Conexión datagrama:
    - **`Connector.open("datagram://www.webyu.com:9000")`**
  - Conexión puerto serie:
    - **`Connector.open("comm:0;baudrate=9600")`**
  - Acceso a ficheros:
    - **`Connector.open("file:/miFichero.txt")`**

# GCF – Clase `HTTPConnection`

- Extiende **`ContentConnection`**
- Estados de la conexión
  - *Setup*
  - *Connected*
  - *Closed*
- Transición de *Setup* a *Connected* motivada por cualquier método que requiera enviar o recibir datos del servidor.
- El interfaz **`HTTPConnection`** proporciona una serie de métodos para cada uno de los estados anteriores.



# GCF – Clase `HTTPConnection`

- *Setup*:
  - **`setRequestMethod`**
  - **`setRequestProperty`**
- *Connected*:
  - **`openInputStream`**
  - **`openOutputStream`**
  - **`openDataInputStream`**
  - **`openDataOutputStream`**
  - **`getLength`**
  - **`getType`**
  - **`getEncoding`**
  - **`getHeaderField`**
  - **`getResponseCode`**
  - **`getResponseMessage`**
  - **`getHeaderFieldInt`**
  - **`getHeaderFieldDate`**
  - **`getExpiration`**
  - **`getDate`**
  - **`getLastModified`**
  - **`getHeaderField`**
  - **`getHeaderFieldKey`**
- *Closed*:
  - **`close`**
  - **`getRequestMethod`**
  - **`getRequestProperty`**
  - **`getURL`**
  - **`getProtocol`**
  - **`getHost`**
  - **`getFile`**
  - **`getRef`**
  - **`getPort`**
  - **`getQuery`**

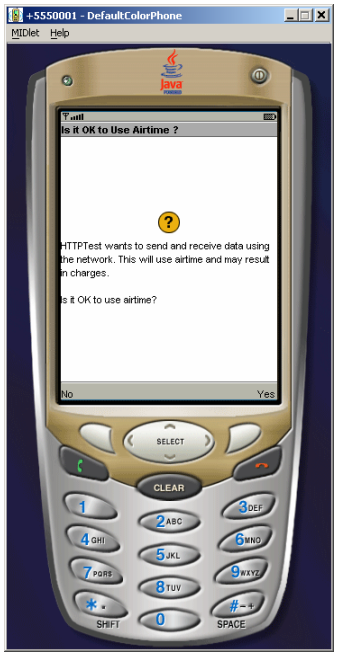
# GCF- Ejemplo HTTPConnection

```
private void download (String url) throws IOException {
   StringBuffer b = new StringBuffer();
   InputStream is = null;
   HttpConnection c = null;
   TextBox t = null;

   try {
     long len = 0 ;
     int ch = 0;
     c = (HttpConnection)Connector.open(url);
     is = c.openInputStream();
     while ((ch = is.read()) != -1) {
       b.append((char)ch); }
     t = new TextBox("Hola...", b.toString(), 1024, 0);
   } finally {
   if (is != null) is.close();
   if (c != null) c.close();
   }
```



# GCF- Ejemplo HTTPConnection



Ver código: **`HTTPTest.java`** – **`HTTPExample.zip`** (proyecto WTK)